

AI-powered Development Enhancing Development Workflows with AI Positron

Octavian Nadolu, Syncro Soft

octavian_nadolu@oxygenxml.com

@OctavianNadolu

© 2025 Syncro Soft SRL. All rights reserved

Agenda

- AI for Development (XSLT, XQuery, Schematron, XSD, and JSON Schema)
- Create Your Own Custom AI Actions
- Oxygen XPath AI Functions
- AI in Schematron
- AI in XSLT/XQuery and Refactoring
- Advantages and Challenges



Oxygen AI Positron

Provides an advanced solution that utilizes artificial intelligence support for editing and developing documents in Oxygen Editor/Developer/Author.

- Content Suggestion and Auto-completion
- Intelligent Contextual Assistance
- Error Detection and Correction
- Documentation and Code Insights
- Personalization and Adaptability



https://www.oxygenxml.com/ai_positron_assistant.html

AI for Development

- AI can be used for developing XSLT, XQuery, Schematron, XSD, and JSON Schema
- Built-in actions
 - Explain code
 - Chat about code
 - Document code
 - Annotate code
 - Generate code
 - Suggest refactoring

WITH AI-POWERED XSLT, XSD, AND SCHEMATRON

**Revolutionize Your
XML Development**

Explain Code

- Generates an explanation of the code
- Available for XSLT, Schematron, XSD, CSS, XQuery, and JSON Schema
- Useful when you want to understand a code

Explain Code in Schematron

- Explain the Schematron code

Explain Code

Explain the code in the chat window

```
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
  <sch:pattern id="NumberOfcharactersInAnAbbreviation">
    <sch:rule context="BBB">
      <sch:report test="string-length(@bbb) &lt; 2">There is not enough letters in the
        abbreviation</sch:report>
      <sch:report test="string-length(@bbb) > 3">There is too much letters in the abbreviation
        </sch:report>
    </sch:rule>
  </sch:pattern>
</sch:schema>
```



In this code snippet, a rule is defined within a pattern to validate the length of an abbreviation denoted by the attribute `@bbb` within an element `BBB`. Here's a breakdown of the code:

- The schema is defined with the namespace `http://purl.oclc.org/dsdl/schematron`.
- A pattern with the id `NumberOfcharactersInAnAbbreviation` is defined.
- Within this pattern, a rule is specified with the context set to `BBB`, which means this rule will be applied to elements named `BBB`.
- Two reports are defined within the rule:
 - The first report checks if the string length of the attribute `@bbb` is less than 2 and provides a corresponding error message.
 - The second report checks if the string length of the attribute `@bbb` is greater than 3 and provides a corresponding error message.

Document Code

- Generates the documentation for the code and includes it as a comment in the document
- Available for XSLT, XSD, and Schematron
- Useful when you want document different parts from your code

Document Code for XSL

- Generate the documentation for XSLT

```
<xsl:template match="timeResolution" mode="header">  
  <xsl:call-template name="listHeader">  
    <xsl:with-param name="name" select="'Time Resolution'"/>  
  </xsl:call-template>  
</xsl:template>
```



Document Code

Create documentation for the code and include it as a comment in the document.

```
<!-- This XSLT code template matches "timeResolution" in header mode and  
calls the "listHeader" template with the parameter "Time Resolution". -->
```


Annotate Code

- Generates the documentation in the entire code
- Available for XSLT, XSD, and Schematron
- Useful when you want to document all your code pieces automatically

Annotate Code for Schematron

- Adds comments to the content

Annotate Code

Adds comments to the content to explain its functionality

```

<sch:rule context="t:Type[text() = 'Doubles']">
  <sch:assert test="../t:Participants/@nbrParticipants mod 2 = 0">
    If you're playing doubles the number of participants must be divisible by 2. </sch:assert>
  <sch:assert test="../t:Participants/@nbrParticipants =
    ../t:Teams/@nbrTeams * 2">If you're playing doubles the number
    of participants must equal the number of teams x 2. </sch:assert>
</sch:rule>
  
```



```

<!-- Rule for double matches -->
<sch:rule context="t:Type[text() = 'Doubles']">
  <!-- Assert that the number of participants must be divisible by 2 for double matches -->
  <sch:assert test="../t:Participants/@nbrParticipants mod 2 = 0">
    If you're playing doubles the number of participants must be divisible by 2. </sch:assert>
  <!-- Assert that the number of participants must equal twice the number of teams for double matches -->
  <sch:assert test="../t:Participants/@nbrParticipants =
    ../t:Teams/@nbrTeams * 2">If you're playing doubles the number
    of participants must equal the number of teams x 2. </sch:assert>
</sch:rule>
  
```

Chat About Code

- Creates a new chat to start a discussion with the AI regarding the code
- Available for XSLT, Schematron, XSD, CSS, XQuery, and JSON Schema
- Useful when you want to improve the code, modify the generated content, or ask questions about the code

Chat About Code in XSD

Chat About Code

Start a discussion about code in the chat window

```
<xs:sequence>
  <xs:element ref="p:name"/>
  <xs:element ref="p:email" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element ref="p:link" minOccurs="0" maxOccurs="1"/>
  <xs:element ref="p:url" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
```



```
<xs:sequence>
  <xs:element ref="p:name"/>
  <xs:element ref="p:email" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element ref="p:link" minOccurs="0" maxOccurs="1"/>
  <xs:element ref="p:url" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="note">
    <xs:simpleType>
      <xs:restriction base="xs:string"> <xs:maxLength value="50"/> </xs:restriction>
    </xs:simpleType>
  </xs:element>
</xs:sequence>
```

Chat About Code

Add a new element note,
that contains less than 50 characters


Generate Code

- Generates the code for the current editor based on the given instructions
- Available for XSLT, Schematron, XSD, CSS, XQuery, and JSON Schema
- Useful when you want to generate new XSLT templates, new Schematron rule, or new elements in an XSD.

Create Schematron using AI

Generate code


An assert that verifies the number of words to be 10



```
<sch:assert test="count(tokenize(., '\s+')) = 10">  
    There should be exactly 10.</sch:assert>
```

Generate code

An assert that verifies if there is an email in text



```
<sch:assert test="matches(., '\b[A-Za-z0-9._%+-]+\@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b')">  
    There is no email in the text</sch:assert>
```

Generate Code for XSD

- Generate XSD code

Generate code

A complex type that defines a purchase order

```
<xs:complexType name="PurchaseOrderType">
  <xs:sequence>
    <xs:element name="CustomerName" type="xs:string"/>
    <xs:element name="OrderDate" type="xs:date"/>
    <xs:element name="Items" type="ItemsType"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ItemsType">
  <xs:sequence>
    <xs:element name="Item" type="ItemType" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ItemType">
  <xs:sequence>
    <xs:element name="ProductName" type="xs:string"/>
    <xs:element name="Quantity" type="xs:integer"/>
    <xs:element name="Price" type="xs:decimal"/>
  </xs:sequence>
</xs:complexType>
```



Generate Code for JSON Schema

- Generate JSON Schema code

```
"phone": {
  "title": "Phone",
  "type": "object",
  "properties": {
    "countryCode": {
      "type": "string",
      "pattern": "^\\+?[1-9]\\d{1,3}$",
      "description": "The international dialing code, e.g., +1 for USA."
    },
    "areaCode": {
      "type": "string",
      "pattern": "^\\d{1,5}$",
      "description": "The area code within the country."
    },
    "number": {
      "type": "string",
      "pattern": "^\\d{4,10}$",
      "description": "The local phone number."
    }
  },
  "required": [
    "countryCode",
    "areaCode",
    "number"
  ],
  "additionalProperties": false
}
```

Generate code

A type that defines a phone



Suggest Refactoring

- Generates a suggestion for refactoring the selected code to simplify it and make it easier to read and understand.
- Available for XSLT and XSD
- Useful when you want to improve your code, to make it modular, and to use new techniques

Start a new Chat

- Retrieval-Augmented Generation (RAG)
- Enable writing content in project
- Project read/write sandbox

Creating Custom Actions

- Framework that helps you to create custom action
 - Validation based on JSON Schema
 - Visual editing
 - Place holders
- Add folder that contains custom actions

Create New XSLT AI Action

- Action that creates a new XSLT file based on a specified content

```

{
  Action ID:          create.new.xslt
  Action name:       Create New XSLT
  Action type:       create-new-document
  Action prompt (context): # CONTEXT #
  You will act as a senior XSLT developer.

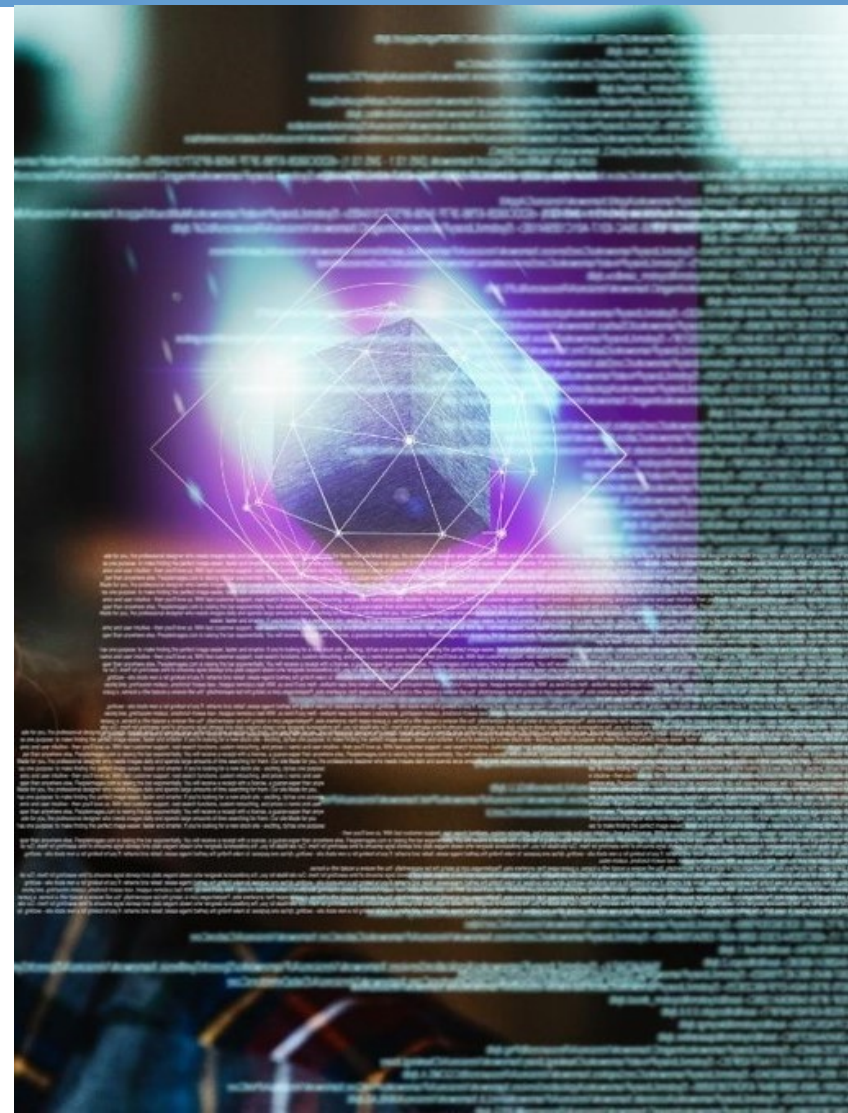
  # OBJECTIVE #
  You are tasked with creating an XSLT starting from the provided text. Create an XSLT with version 3.0

  # RESPONSE #
  Respond with just the XSLT file content, without any other explanations.

  ▾ Advanced configuration parameters:
    Functions to be used:
    ▾ get_content_for_document_url
    New function
}
  
```

Automate AI Actions

- Use the AI engine API to perform complex actions
- Automate the process



AI XPath Functions

- Functions can provide a specific built-in prompt
 - `ai:verify-content(instruction, content)`
“You are a technical writer and you need to verify the following and respond with true or false:” + Is active voice used in the description? + content
 - `ai:transform-content(instruction, content)`
“You are a technical writer and you need perform the following task:” + Rephrase to use active voice + content
 - `ai:invoke-action(actionID, instruction*, content)`
“You need perform the following task:” + generate.indexterms (action prompt) + Do not add other explanations + content

AI Functions Usage

- Use the AI functions from XSL and XQuery
 - Transform content using AI
 - Create refactoring actions based on AI
- Use the AI functions from Schematron and SQF
 - Verify document content automatically using AI
 - Correct problems in document using AI

Transform content using XSL and AI

- Generate the documentation for all templates and functions

```
<xsl:template match="xsl:template | xsl:function">
  <xsl:comment><xsl:value-of select="ai:transform-content(
    'As a developer, create a single phrase of documentation for the provided XSL content.', .)"/>
  </xsl:comment>

  <xsl:copy>
    <xsl:apply-templates select="node() | @*"/>
  </xsl:copy>
</xsl:template>
```

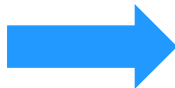

Transform content using XSL and AI

```
<xsl:template match="/">
  <html>
    <head><title>Employees</title></head>
    <xsl:element name="table">
      <xsl:attribute name="border">1</xsl:attribute>
      <tr class="header">
        <xsl:attribute name="bgcolor">#336666</xsl:attribute>
        <xsl:attribute name="align">center</xsl:attribute>
        <td>
          <font face="Arial" size="3">
            <b>Name</b>
          </font>
        </td>
      </tr>
    </xsl:element>
  </html>
</xsl:template>

...

<xsl:template match="//p:person">
  <xsl:element name="tr">
    <xsl:attribute name="align">center</xsl:attribute>
    <xsl:element name="td">
      <xsl:attribute name="width">120</xsl:attribute>
      <font face="verdana" size="3">
        <i>
          <xsl:value-of select="p:name/p:family/text()"/>
          <xsl:text> </xsl:text>
          <xsl:value-of select="p:name/p:given/text()"/>
        </i>
      </font>
    </xsl:element>
  </tr>
</xsl:template>

...
```



```
<!--This template styles and displays an "Employees" table with
columns for "Name", "Email", and "Link".-->
<xsl:template match="/">
  <html>
    <head><title>Employees</title></head>
    <xsl:element name="table">
      <xsl:attribute name="border">1</xsl:attribute>
      <tr class="header">
        <xsl:attribute name="bgcolor">#336666</xsl:attribute>
      </tr>
    </xsl:element>
  </html>
</xsl:template>

...

<!--This XSLT template transforms `p:person` elements into
centered HTML table rows (`tr`) with columns (`td`) for the
person's full name, email, and link attributes for subordinates
and manager, formatted with Verdana font.-->
<xsl:template match="//p:person">
  <xsl:element name="tr">
    <xsl:attribute name="align">center</xsl:attribute>
    <xsl:element name="td">
      <xsl:attribute name="width">120</xsl:attribute>
      <font face="verdana" size="3">
        <i>
          <xsl:value-of select="p:name/p:family/text()"/>
          <xsl:text> </xsl:text>
          <xsl:value-of select="p:name/p:given/text()"/>
        </i>
      </font>
    </xsl:element>
  </tr>
</xsl:template>

...
```

AI Complex Interactions

- Use AI XPath functions to create complex interactions

```
ai:transform-content(system, (user, assistant,)* user)
```

- **system**: “You are a marketing specialist. Create 3 variants of marketing post” |
- **user**: “Oxygen AI Positron service uses the OpenAI platform to help technical documentation writers with features like document generation ...”
- **assistant**: AI response – *3 variants of post*
- **user**: “For each variant, enumerate 3 strong points and 3 disadvantages”
- **assistant**: AI response – *strong points and disadvantages*
- **user**: “Based on the advantages and disadvantages, choose the best variant”
- **Assistant**: AI response – *best marketing post*

Transform content using XSL and AI

- `ai:transform-content(instruction, (user, agent,)* content)`

```
<!-- Step 1: rephrase in 3 possible ways -->
```

```
<sch:let name="user-3variants" value="Provide 3 variants of rephrasing the given text in strictly less than 75 words."/>
```

```
<sch:let name="assistant-3Variants" value="ai:transform-content($user-3variants, $currentShortDesc)"/>
```

```
<!-- Step 2: get the advantages and disadvantages of each variant -->
```

```
<sch:let name="user-compare3varinats" value="For each of the given text variants, enumerate 3 strong points and 3 weak points about how it is written."/>
```

```
<sch:let name="assistant-AdvAndDisadv" value="ai:transform-content($user-compare3varinats, $assistant-3Variants)"/>
```

```
<!-- Step 3: ask the AI to choose the best variant -->
```

```
<sch:let name="system-bestVariant"
```

```
  value="Based on the strong points and weak point of the way each text is written, choose the one of them that best summarizes the following text and is the best written. Respond with just the best text, nothing else."/>
```

```
<sch:let name="assistantBestVariant" value="ai:transform-content($system-bestVariant,  
  $user-3variants, $assistant-3Variants, $user-compare3varinats, $assistant-AdvAndDisadv,  
  $currentShortDesc)"/>
```

AI in XSL and XQuery

- Advantages of using AI with XSL and XQuery
 - Perform refactoring actions
 - Use AI to generate the content
 - Control the content that is sent
 - Control the content that is modified
 - Automate the process
- Challenges
 - Cost for transformation
 - The response from the AI server is not instant
 - Responses can sometimes be inaccurate

Verify content using Schematron

- Example of a rule that checks if the text uses active voice

In the description we should use active voice

`shortdesc` **Short Description:** The journey into the world of AI is continued through the exploration of its application in conjunction with Schematron and Schematron Quick Fix (SQF) for content verification and correction. In this webinar, a comprehensive overview of AI will be offered, the potential advantages it brings will be highlighted, and the challenges encountered when utilizing AI for these purposes will be illuminated. `shortdesc`



Check the text voice

- Rule that verifies if the text voice is active

```
<sch:rule context="shortdesc">  
  <sch:assert test="ai:verify-content('Is active voice used?', .)">  
    In the description we should use active voice.</sch:assert>  
</sch:rule>
```

Correct the text voice

- Example fix that reformulates the text to use active voice

Reformulate the text to use active voice

Short Description: The journey into the world of AI is continued through the exploration of its application in conjunction with Schematron and Schematron Quick Fix (SQF) for content verification and correction. In this webinar, a comprehensive overview of AI will be offered, the potential advantages it brings will be highlighted, and the challenges encountered when utilizing AI for these purposes will be illuminated.



Short Description: Explore the application of AI in conjunction with Schematron and SQF for content verification and correction in the webinar. Offer a comprehensive overview of AI, highlight its potential advantages, and illuminate the challenges encountered when utilizing AI for these purposes.

Correct the text voice

- SQF fix that reformulates the text to use active voice

```
<sqf:fix id="rephrase">
  <sqf:description>
    <sqf:title>Reformulate the text to use active voice</sqf:title>
  </sqf:description>
  <sqf:replace match="text()" select="ai:transform-content('
    Reformulate to use active voice', .)"/>
</sqf:fix>
```


Use AI to Create Short Description

- Add a short descriptive element. Generate the text from the current document content using AI

```
<sqf:add match="title" position="after">
  <shortdesc>
    <xsl:value-of select="
      ai:transform-content('You are a technical documentation writer.
      Generate a short description as text in less than 30 words for this content:',
      string-join(parent::*//text(), "))/>
  </shortdesc>
</sqf:add>
```

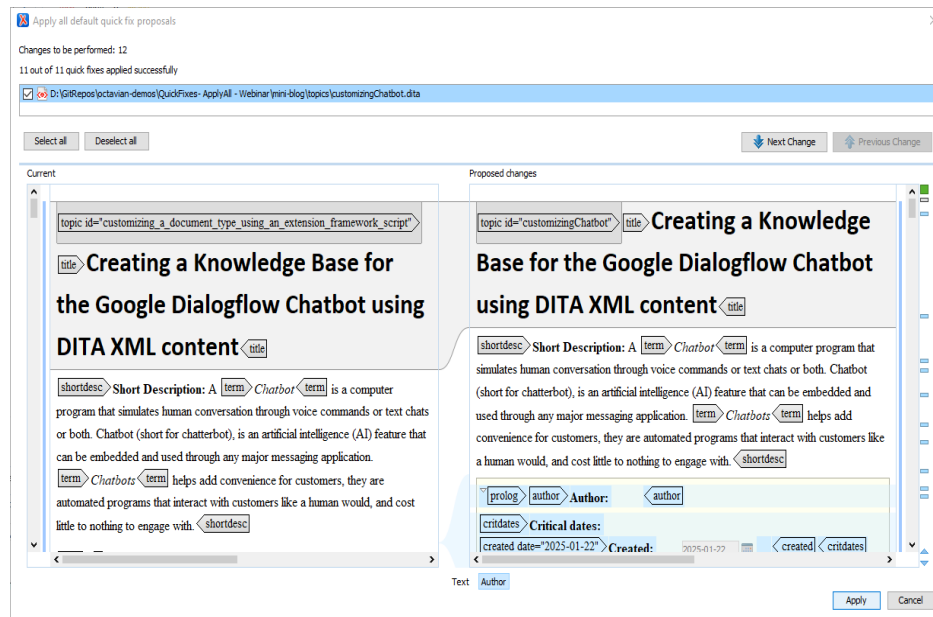
Use AI to Create Text from Image

- Add an alternate element. Generate the text by reading the image using AI

```
<sqf:add node-type="element" target="alt">  
  <xsl:value-of select="  
    ai:transform-content(  
      'Create a short alternate text description for this image:',  
      concat('${attach(', resolve-uri(@href, base-uri()), ')}')"/>  
</sqf:add>
```

Apply All Quick Fixes

- Efficiency: Manual correction is time-consuming
- Batch Resolution: correct all validation issues in a document
- Preview and Confirmation: Check the modifications before they are applied
- Page Flexibility: Works in both Text and Author modes



AI and Schematron+SQF

- Advantages of using AI with Schematron and SQF
 - Verify and correct your documents using AI
 - Define the instructions to be sent to the AI engine
 - Control the content that is sent
 - Control the content that is modified
 - Automate the content verification and correction
- Challenges
 - High cost for validation as you type or multiple validations
 - The response from the AI server is not instant
 - Responses can sometimes be inaccurate

Conclusion

- AI constantly growing and improving
- Use AI for developing your schemas or stylesheets
- Use AI XPath functions to automate the process
- Refactor your content using AI
- Do not expect to do your job, use AI to improve your job



Future Plans

- New development actions
- Improve the editing for custom actions
- Support for testing the prompts
- *feedback is welcome*



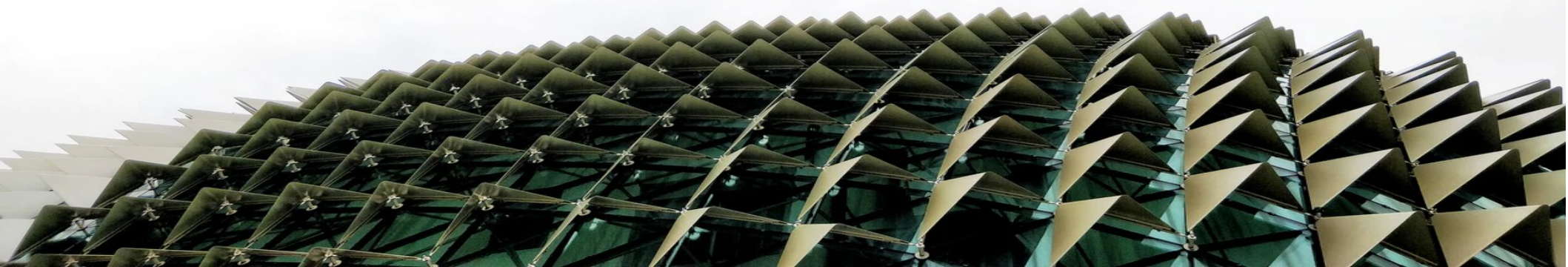
Question: What future developments would you like to see?

- ❑ New development actions
- ❑ Improve the editing for custom actions
- ❑ Support for testing the prompts
- ❑ Other (use the question panel)



Resources

- oxygenxml.com/doc/ug-addons/topics/ai_positron.html
- github.com/oxygenxml-incubator/ai-positron-assistant-samples
- <https://platform.openai.com/docs/guides/chat>
- <http://schematron.com/>
- <http://schematron-quickfix.github.io/sqf>





Questions?

Octavian Nadolu
Project Manager at Syncro Soft

octavian.nadolu@oxygenxml.com

Twitter: [@OctavianNadolu](https://twitter.com/OctavianNadolu)

LinkedIn: [octaviannadolu](https://www.linkedin.com/in/octaviannadolu)